DAG 分割模型下的云工作流调度策略

薛　凡

(黄淮学院 创新创业学院，河南 驻马店 463000)

摘　要： 工作流调度是工程管理领域中经济有效的优化手段，为了优化云工作流调度的经济代价和执行效率，提出一种基于有向无循环图 DAG 分割的工作流调度算法 PBWS。以工作流调度效率与代价同步优化为目标，算法将调度求解过程划分为三个阶段进行：工作流 DAG 结构分割、分割结构调整及资源分配。工作流 DAG 结构分割阶段在确保任务间执行顺序依赖的同时求解初始的任务分割图；分割结构调整阶段以降低执行跨度为目标，在不同分割间对任务进行重分配；资源分配阶段旨在选择代价最高效的任务与资源映射关系，确保资源的总空闲时间最小。利用五种科学工作流 DAG 模型对算法进行了仿真实验。结果表明。PBWS 算法仅以较小的执行跨度为开销，极大降低了工作流执行代价，实现了调度效率与调度代价的同步优化，其综合性能是优于同类型算法的。

关键词： 云计算；科学工作流；调度优化；DAG 分割；执行跨度

中图分类号： TP393　　doi: 10.3969/j.issn.1001-3695.2018.04.0375

Cloud workflow scheduling strategy in DAG partition model

Xue Fan

(*Innovation & Entrepreneurship College*, *Huanghuai University*, *Zhumadian Henan* 463000, *China*)

Abstract: Workflow scheduling is a kind of economical and effective method in engineering management area. For optimizing the economical cost and scheduling efficiency of cloud workflow scheduling, a workflow scheduling algorithm based on Directed Acyclic Graph DAG partition PBWS is proposed. With the goal of optimizing synchronously the workflow scheduling efficiency and cost, our algorithm divides the scheduling solution into three stages: DAG partition of the workflow structure, partition structure adjustment and resource allocation. DAG partition of the workflow structure is to get the initial tasks partition graph when guarantees the execution order-dependency between tasks, the partition structure adjustemnt is to re-allocate tasks in different partitions with a goal of reducing execution makespan and the resource allocaiton is to determine the most cost-efficient mateches between tasks and resources ensuring the minization of the total idel time of resource. We construct some simulation experiments for algorithms by the five types of scientific workflow DAG model. The experimental results show PBWS can greatly reduce the execution cost of wrokflow in terms of cost by a large margin with litter overhead on execution makespan and realize the synchronous optimization of the scheuduling efficiency and the scheduling cost, whose overall performance performs better than the same type of algorithms.

Key words: cloud computing; scientific workflow; scheduling optimizatin; directed acyclic graph partition; execution makespan

0　引言

云数据中心可以提供巨大的 IT 能力，服务于社交、邮件、金融和工业等诸多领域。尤其在科学计算和工程控制应用中，爆炸式、复杂大规模的数据增长对计算环境提出了更高的要求。该类领域中的多数应用任务通常体现为工作流（workflow）形式[1]，其结构可参见如图 1 所示的五种常规科学领域的工作流。工作流中，每个任务作为工作流整体的一部分，或者作为初始数据的传送者，或者作为中间数据的中继者，进而形成一种任务间的顺序依赖，协作完成一种应用任务。

工作流调度本质上是 NP 问题，多数工作通过设计单目标函数进行优化。例如在文献[2~4]中，作者设计调度算法以寻找满足预先定义的工作流执行截止期限的最低价调度方案。比较而言，在文献[5~8]中，所设计的算法则是寻找满足预先定义的预算的最快调度方案。相比较而言，文献[9~11]则是以降低执行时间为目标，而未考虑费用约束问题。虽然以上方法引入了预算或截止期限约束，但总体来说，算法优化均是单目标优化为限制的，未能很好地实现符合云环境资源使用特征的工作流调

收稿日期： 2018-04-19；**修回日期：** 2018-06-07

作者简介： 薛凡（1978-），男，河南驻马店人，博士研究生，讲师，主要研究方向为智能计算、云计算（xuefan78@sohu.com）.

度优化。

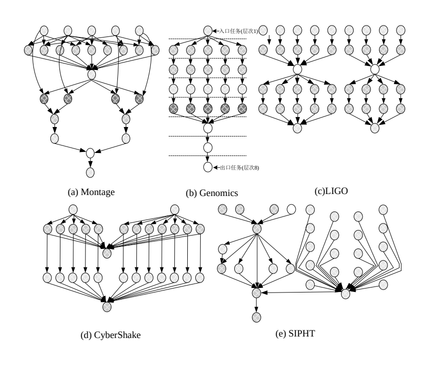


图 1　科学工作流结构

本文同步考虑执行跨度和执行代价的同步最小化，并设计一种基于 DAG 分割的工作流调度算法。算法将工作流 DAG 结构划分为包括多个任务的多个群组（分割 partition），进而实现分割与云资源动态能力的优化匹配。算法通过三阶段的调度优化，最终可以实现工作流调度效率与代价的均衡。

1　系统模型

工作流表示为一个有向无循环图 DAG G=<V,E>，其中 V 为任务集，E 为边集。每条边连接两个任务，代表任务间的顺序约束。给定资源集合 $R=R_1,R_2,\ldots,R_m$，资源集以 Amazon EC2 的实例类型进行配置，包括 m4.xlarge、c4.2xlarge 和 r3.4xlarge。每个资源集 $R_i \in R$ 由同质资源构成，配置不同数量的处理器内核、处理能力、内存和存储空间。基于资源能力对集合 R 进行排序，若 $j>i$，则属于集合 R_i 中的资源拥有低于集合 R_j 的资源能力。仅当接收父任务产生的数据后，子任务 v_i 才可以开始执行。在所有父任务中，最迟发送数据的任务称为最有影响父任务 MIP。令 $c_{(i,j)}$为任务 v_i 与 v_j 间的通信代价，工作流的完成时间可定义为执行跨度 makespan。

对于任务 $v_i \in V$ 的最早开始时间 EST 和最早完成时间 EFT 可定义为

$$EST(v_i)=\begin{cases}0, v_i\text{为入口任务}\\ EFT(MIP(v_i))+c_{MIP(v_i),i}, \quad \text{否则}\end{cases} \tag{1}$$

$$EFT(v_i)=EST(v_i)+w_i \tag{2}$$

其中：w_i 为任务 v_i 的执行时间。当调度至相同资源上的另一个任务的实际完成时间晚于 EST(v_i)时，任务的实际开始时间 AST 和实际完成时间 AFT 不同于 EST 和 EFT。

工作流的调度跨度 makespan 主要受关键路径上任务的执行时间的影响，该路径拥有最高的代价（通信和计算代价），并结束于出口任务。任务 $v_i \in V$ 的计算代价（执行时间 w_i）取决于调度任务的资源能力。资源能力越高，执行时间越短。令 w_i^j 为计算代价（时间），其中，i 为任务 $v_i \in V$ 的索引，j 为资源集 $R_j \in R$ 的索引。为了简化计算过程，为每个资源集合 $R_i \in R$ 引入一个常量因子 cv_i>0。给定任务 v_i，每个资源集合常量 cv_i（$R_i \in R$）代表任务的近似相对执行时间。例如，若 cv_3=4，则任务的执行时间 w_i^3=(1/4)w_i^1。令 t_i 为每小时利用资源集 $R_i \in R$ 中资源的代价，该值取决于资源能力，若 $i<j$，则 $t_i<t_j$。

工作流调度目标即为分配给定工作流的任务至 R 中资源集 S，同时确保资源利用总代价和任务执行跨度最小化。

2　算法设计

2.1　算法概述

PBWS算法的目标是分割工作流任务形成任务群组，进而产生有效的资源与任务间的映射，分割目标是简化资源分配过程。基于任务分割区域间的数据依赖性，决定分配至每个分割的资源能力，并确保工作流执行跨度与代价最小化。PBWS算法由三个步骤组成：a）DAG分割步骤；b）分割调整步骤；c）资源分配步骤。

DAG分割步骤的目标是划分给定的工作流DAG结构，使得得到的DAG分割在确保任务间依赖的同时，简化资源分配过程，而在后续步骤中，每个分割均视为单个节点对待。分割调整步骤中，为了进一步降低执行跨度，某些任务需要重新分配并重新划分至不同分割中。资源分配步骤的目标是实现任务与资源间代价最有效的映射关系，确保资源总空闲时间最小化。

2.2　DAG 分割步骤

算法1是DAG分割步骤的伪代码。算法中，首先需要在考虑工作流的关键路径CP的执行时间情况下决定分配至每个分割的任务数量。由于关键路径上的任务执行时间之和（即关键路径长度，表示为l(CP)）代表执行跨度makespan的下限（最优解），DAG分割需要确保属于相同分割中的任务的总执行时间小于或等于l(CP)（步骤8中的while循环）。由于处于关键路径CP上任务被调度至同一资源上，所以关键路径长度仅包括执行时间，而在任意其他分割（算法1中的P’）中的任务调度长度则包括执行时间和通信时间。分割P的长度（即代价）定义为

$$l(P)=\sum_{i\in P} w_i^1+\sum_{e(v_i,v_j)\in P} c_{(i,j)} \tag{3}$$

其中：等式右边第一项为分割 P 中的任务在最低价资源 w^1 上的计算代价之和；第二项为分割 P 中任务间的通信代价之和。

DAG 分割步骤从工作流结构中最低层次的任务开始执行（如图 1（b）中的工作流不同层次），该步骤添加任务至当前构造的分割中，直到在不违背 l(CP)限制的情况下没有新任务添加进来为至，即步骤 12。一旦属于相同层次的所有任务分配至分割中，处于下一层次（高层次）的任务继续进行分割，直到所有任务被分配至不同分割中为止。

算法 1　DAG 分割过程

```
1.  Procedure Partitioning(G,l(CP))
2.     Input:G=<V,E>,l(CP)
3.     Output:P=<p1,p2,…>
4.     ↘(collection of partitions)
5.     N←order tasks based on level
6.     while N is not empty do
7.       P'←∅
```

```
        while l(P')<l(CP) do
            add first task in N to P'
            remove the task from N
        end while
        if l(P')>l(CP) then
            undo last step
        end if
        add P' to P
    end while
    return P
end procedure
```

2.3　分割调整步骤

为了确保 DAG 分割步骤中产生的任务分割在最终资源分配过程中拥有最优的粒度，不同分割的任务需要重新分配调整。寻找最优的分割主要涉及每个分割中的任务数量及其分配的资源能力。

由于任务分割与任务间的数据依赖存在相互关系，不同分割间的任务重新分配时需要涉及 EST 和 EFT 时间值的重新计算。对于每个任务，计算两个值分别为 s_{est} 和 p_{est}。任务 v_i 的 s_{est} 值表示 v_i 的子任务的平均 EST，且子任务与 v_i 不在同一分割中。任务 v_i 的 p_{est} 值表示与 v_i 同属一个分割中的任务的平均 EST。基于这两个值，对于每个任务 $v_i \in V$，可以决定是否通过将 v_i 重新分配至另一分割（拥有最小的 EST(v_i)）从而调整 v_i 的分割以降低执行跨度。该步骤可以通过寻找 $s_{est} \le p_{est}$ 的任务完成。这种分割调整对执行跨度和执行代价性能具有重要影响，它可以降低由于处于不同分割的父任务的执行导致其子任务的等待时间。如果重新分配的任务在其分割内拥有子任务，则其子任务也需考虑重新分配至新的分割。仅当其执行时间小于任务所在分割中任务的平均执行时间时，该任务才考虑需要重新分配。通过该标准，可以确保任务在重新分配的分割中尽早执行。

将分割调整步骤得到的结果表示为 DAG $G'=<V',E'>$，其中 V'表示分割集合，E'表示分割间的数据依赖集合。任意两个分割 $v'_i, v'_j \in V'$通过一条以上的有向边连接，每条有向边 $e'(v'_i, v'_j) \in E'$表示任务 $v_p \in v'_j$和其在 v'_i中的父任务。同时，边展示了属于 v'_j的任务的最小 EST，即 e_{est}。

2.4　资源分配步骤

算法 2 给出了资源分配步骤的伪代码。该步骤包括资源集合识别和资源分配两个阶段。资源集合识别阶段中，需要决定分配至每个分割的资源集合类型($i, R_i \in R$)。资源分配阶段中，每个任务 v_j 分配至资源 $r \in R_i$，使得 r 属于与 v_j 所处分割相同的资源集合类型。资源集合识别阶段的目标是寻找分配至分割的资源集合类型，使得分割间的数据依赖关系达到最小化。资源分配阶段的目标则是基于松驰参数 β 和所识别的资源集合类型决定最快的调度方案。

算法详细说明：资源集合识别阶段开始于将分割划分为不同的分割群组，即步骤 4。一个分割群组包括一个群组领导 $P_i \in P$ 和与 P_i 连接的父分割。拥有一个以上依赖关系的分割可属于多个群组中，那么，对于每个群组（步骤 8 中的 for 循环），从拥有出口分割的群组开始作为一个领导群组（出口群组即包括出口任务的群组），属于该群组的分割（除领导以外）开始识别其资源集合类型（步骤 9 的 for 循环）。在每个群组中，每个分割的资源集合类型($i, R_i \in R$)取决于群组领导的 e_{est} 值。为了决定资源集合类型，每个分割需要识别其执行开始时间，即 EST，将该时间记为 e_t。此外，每个分割需要计算其任务的计算时间 c_{Pi}。获得这两个值后，将每个分割的 e_t+c_{Pi} 除以其群组领导的 e_{est} 值，即步骤 10。该除法结果表示为了降低执行跨度该分割要求执行时的计算时间降低量。该降低量可以通过将与该除法结果最相近的 c_{vi} 的资源集合类型至该分割来得到，即步骤 11。如果任一分割均包含单个分割，则该分割被分配至最低资源集合类型 R_1。一旦确定的一个分割的资源集合类型，属于该分割中的每个任务的 EST 和 EFT 即可重新计算。在该分割中的子任务的 EST 和 EFT 同时被重新计算，即步骤 12～14。分割识别其资源集合类型后，如果该分割是一个群组领导，则它将告之其群组成员/分割开始执行资源集合识别阶段。当一个分割属于多个群组时，该分割被分配至最高的资源集合类型。

所有分割确定其资源集合类型后，资源分配阶段基于每个分割所识别的资源集合类型开始向任务分配资源，即步骤 17～20。该阶段中，基于松驰参数 β 的值，确定每个任务的 AST 和 AFT。参数 β（$0 \le \beta \le 1$）用于控制执行任务的资源数量。增加该参数值可以降低资源使用的数量，此时相比执行跨度将有利于执行代价的优化。而降低该参数值，将有利于优化执行跨度。该阶段开始于计算每个任务 AST 的上限 UB 和下限 LB。两个值代表每个任务执行时所允许的延时。每个任务的 AST 根据参数 β 进行设置，即步骤 18。如果 $\beta=1$，则任务 AST 设置为 UB，且算法利用最少数量的资源；如果 $\beta=0$，则任务 AST 设置为 LB，且算法利用最多数量的资源。任务 v_i 的 LB 为$(MIP_i)+c_{(MIP_{i,i})}$，表示无资源限制时执行任务 v_i 的最早时间。当属于相同分割的任务分配至相同资源时，任务 AST 为其上限 UB。在这种情况下，任务 v_i 的 UB 为 AFT(v_j)+$c_{j,i}$，其中，v_j 为插入分割中的 v_i 间的中间任务，$c_{j,i}$ 任务 v_i 与 v_j 间的通信时间。每次迭代中，拥有最低 AST 的未分配任务考虑为待分配任务，该任务将分配至与任务所处分割相匹配且能保证及时执行的资源集合类型中的资源。如果不存在该资源，则启用资源集合类型中的新实例。

算法 2　资源分配过程

```
1.  Procedure Reallocation((G',P,β,R))
2.      input:G'=<V',E'>,P partitions,
3.          β,R=R1,...,Rm resources
4.      C←group P based on successor partition ID
5.       ↘each group Ci has one or more partitions
6.      order C based on the dependancy between the tasks
7.       ↘Ci,Cj(i<j),Ci depends on Cj data
8.      for each Ci∈C do
```

```
            for each Pj∈Ci do
                h←(er+cPi)/eest
                call AssignResource(h,Pj,R)
                for each vi∈Pj do
                    call UpdateESTEFT(vi,h,G')
                end for
            end for
        end for
        for each vi∈V do
            AST(vi) ←β×(UBi-LBi)+LBi
            AFT(vi) ←AST(vi)+wi
        end for
    end procudure
```

2.5　算例说明

以图 2 对 PBWS 算法的思想进行阐述。为了简化描述，算例中未显示计算和通信代价。图 2（b）是 DAG 分割步骤得到的结果，每个任务被分配至一个分割中；（c）是分割调整步骤得到的结果。该步骤中，任务 t_9 满足拥有最小 EST 条件，被重新分配至分割 p_2。图 2（d）的资源分配步骤中，每个分割（除根分割）标志其 e_r+e_{Pi} 为 H 及其 e_{est} 值。在该算例中，H_1=40，e_{est1}=20，H_2=20，e_{est2}=20。由于 p_3 是分割 p_1 和 p_2 的子分割，这两个分割通过其 H 值与 e_{est} 值相除识别其需要的资源集合类型，这表明 p_1 的资源集合类型为 2(R_2)，p_2 的资源集合类型为 1(R_1)。这表明在 p_1 中的所有任务被分配至 R_2 中的资源，p_2 中的所有任务被分配至 R_1 中的资源上。一旦相同群组中的分割（拥有相同的后代）识别了其资源集合类型，即可获得每个分割中任务的执行次序和每个分割的 e_{est}。然后拥有已识别资源集合类型的领导分割告之其群组成员开始执行资源集合识别阶段。该算例中，p_1 为分割 p_2 的父分割。然而 p_1 并未作改变，由于所识别的资源集合类型 2(R_2)高于 1(R_1)。任务的执行次序取决于所在的层次，如 p_1 中任务的执行次序为 t_{10}、t_7、t_8、t_4。每个分割中任务的 AST 取决于 β 值。换言之，如果分配单个资源至每个分割（β=1），则 p_1 中任务的 AST 分别为 0、10、20、30。该算例中，假设 p_1 中每个任务的计算代价为 10。如果假设 β=0，p_1 中任务的 AST 则分别为 0、10、10、20。此时，t_7 和 t_8 分配至不同的资源上。

2.6　算法时间复杂度分析

PBWS 算法首先需要计算每个任务的 EST 和 EFT，时间复杂度为 $O(|V|)$；然后计算关键路径的时间复杂度为 $O(|V|)$。DAG 分割步骤至多花费 $O(|V|^2)$可以收敛。分割调整步骤中，对于每个任务，计算其子任务的平均 EFT 的时间复杂度为 $O(|V|^2)$。此外，重新分配任务的时间复杂度为 $O(|V|(|P|-1))$，因此，分割调整步骤的时间复杂度总体为 $O(|V|^2+|V|(|P|-1))$。资源分配步骤中，对分割进行排序的时间复杂度为 $O(|P|^2)$。将分割划分为群组的时间复杂度为 $O(|P|)$，资源分配过程的时间复杂度为 $O(|P||R|)$。因此，资源分配步骤的总体时间复杂度为 $O(|P|^2+|P|(1+|R|)$。在最差情况下，DAG 分割数量等于任务数量，即一个任务为一个分割，因此，PBWS 算法的最差时间复杂度为 $O(|V|^2+|P|^2+(|V||P|)+|P||R|)= O(|V|^2+|V||R|)$。

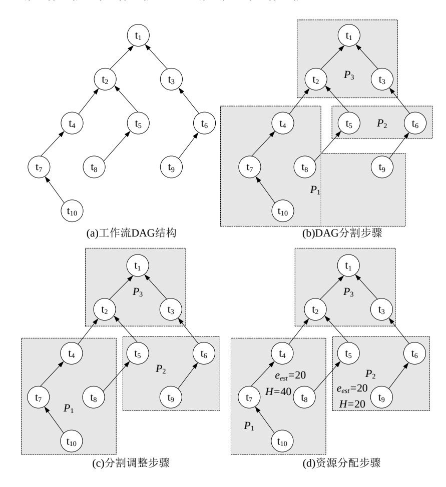


图 2　算例说明

3　仿真实验

本章通过仿真实验分析 PBWS 算法的性能，选择 IC-PCP[4] 和 HEFT[11]两种算法作为基准算法，选择的云计算环境仿真工具为 CloudSim，输入工作流由 Pegasus 工作流发生器得到 (https://confluence.pegasus.isi.edu/display/pegasus/WorkflowGenerator)。利用图 1 所示的五种现实世界中的科学工作流进行实验测试，包括 CyberShake、Epigenomics、LIGO、Montage 和 SIPHT，为每种工作流配置 1 000 个任务。资源集合 R=R_1,R_2,R_3，具体来说，最快资源 R_3 的价格是最慢资源 R_1 的三倍，资源节点间的带宽设置为 1 Gbps，且假设可用的资源数量是不受限制的，β 的取值为(0,0.5,1)。

3.1　性能指标

实验主要观察算法的执行跨度和代价指标。代价包括任务的执行代价、资源初始化代价和数据传输代价。显然，在调度方案中增加资源数量对总体代价具有重要影响。因此，采用标准化代价 N_c 比较该指标性能，该指标考虑了任务执行代价和所使用的资源数量。对于每个调度，将获得的代价除以在价格最低的资源上的调度代价，该值代表相比最低价调度所获得调度方案的代价程度。然后，将该值乘以所获调度方案中所使用的资源数量，即

$$N_c = \frac{c}{s_c} \times r \tag{4}$$

其中：c为资源上执行任务的数量；r为该调度中所使用的资源数量；s_c为任务调度至最低价资源上的代价。

3.2　实验结果

图 3 和 4 是三种算法在执行跨度和标准化代价方面的性能比较。可以看出，在代价上 PBWS 始终优于其他算法，而 HEFT 在执行跨度上优于其他算法，这表明 PBWS 为了降低代价牺牲了部分执行效率。具体地，笔者根据五种工作流的结构特征分成三个部分进行讨论。

a）CyberShake 工作流。对于 CyberShake，当 β 值为 0.5 和 0 时，即算法更倾向于最小化执行跨度时，PBWS 和 HEFT 拥有非常相近的执行跨度（图 3（a）），且仅仅只带来较小的执行代价的增加（图 4（a））。具体地，由于降低 β 值可以导致更多的资源使用，且 CyberShake 工作流（图 1（d））不存在瓶颈任务，PBWS 所构造的多个分割能够并行执行从而降低执行跨度。尽管 IC-PCP 得到的执行跨度是整体最小的，但获得这种更好的性能是以更多的资源使用和更高的代价为代价的（图 4（a））。此外，当 β=0 时 HEFT 拥有更高的代价，这是由于 HEFT 为了得到更高的执行跨度性能利用更高价的资源。

b）Epigenomics 和 LIGO 工作流。对于这两种工作流，PBWS 获得与 IC-PCP 相近的执行跨度（图 3（b）和（c）），但 PBWS 拥有比 IC-PCP 更低的代价（图 4（b）和（c））。对于标准化代价，由于 IC-PCP 获得了与 β 无关的相同执行跨度，所以期望得到的调度也拥有相同的代价。对于 PBWS，得到的执行跨度取决于 β 值和工作流结构。降低 β 值可以增加使用资源的数量，且可影响最终的执行跨度。然而只有在该值的降低使得分配至每个分割中的资源数量增加才会拥有这种效果。两种工作流在结构上拥有一致性，使得在满足数据依赖上可以使分割的执行具有更好的连续性。因此，在这种条件下，改变 β 值并不会对 PBWS 的性能造成巨大影响。实验中设置 HEFT 使用的资源数量与 PBWS 相同。对于 PBWS，改变 β 值对利用的资源数量带来了更小的变化。因此，降低 β 值对于 HEFT 的执行跨度影响很小。但总体而言，HEFT 作为一种贪婪算法，可以获得更小的执行跨度，但比较 PBWS 则拥有更高的代价。

c）Montage 和 SIPHT 工作流。对于这两种工作流，结果表明 IC-PCP 在大多数情况下，其标准化代价均拥有最差的性能，这与工作流的结构相关。在 Montage 中，处于最后的层次中的任务形成了一种管道结构，这对 IC-PCP 具有巨大的负面影响，这主要源于此时最长关键路径的长度在决定算法效率上具有更重要的影响。在 SIPHT 中，由于出现了左侧路径，所以效果与 Montage 类似。

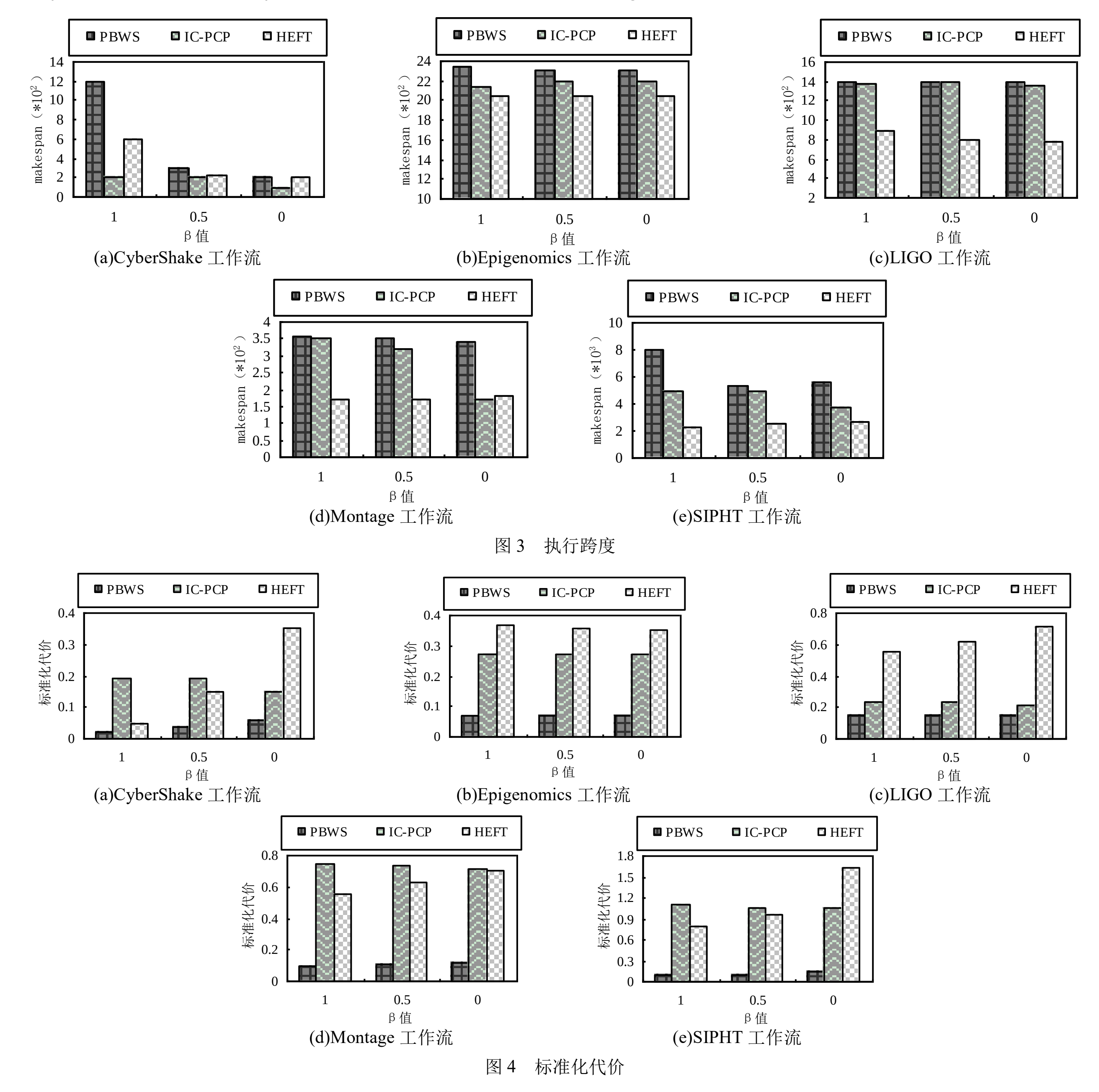


图 3　执行跨度

图 4　标准化代价

表 1 给出算法在两个性能指标上的具体值表现。其结果再次证明 PBWS 得到的调度比较其他算法拥有更低的代价，且仅是以牺牲微小的执行跨度得到的。

表 1　调度性能

工作流类型	算法	执行跨度		标准化代价	
		β=0	β=1	β=0	β=1
CyberShake	PBWS	194.88	1196.56	0.07	0.01
	IC-PCP	107.22	213.90	0.14	0.18
	HEFT	185.22	580.19	0.35	0.06
Epigenomics	PBWS	23089.17	23186.92	0.07	0.07
	IC-PCP	21919.12	21919.12	0.27	0.27
	HEFT	20583.27	20695.50	0.36	0.38
LIGO	PBWS	1394.06	1427.70	0.15	0.15
	IC-PCP	1389.97	1395.13	0.22	0.24
	HEFT	784.01	878.84	0.73	0.57
Montage	PBWS	341.37	370.48	0.13	0.10
	IC-PCP	174.77	349.55	0.73	0.76
	HEFT	180.22	180.86	0.72	0.55
SIPHT	PBWS	5261.95	7893.58	0.08	0.07
	IC-PCP	3563.85	5261.95	1.04	1.09
	HEFT	2630.98	2630.98	1.66	0.82

总体来说，在不同类型的科学工作流结构下，PBWS 的综合性能是较优化的。相比同类型算法，PBWS 为了进行执行效率与代价的同步优化，将优化目标分两阶段进行，前一阶段通过初始任务 DAG 分割和分割结构调整后的任务重分配来提高执行效率，降低执行跨度；后一阶段则在资源映射时优化了执行代价，使得资源空闲时间更少，资源利用更充分。虽然算法在单项指标上无法实现最优，但在多指标的综合性能上可以更加均衡，也更符合云环境的实际调度环境及工作流调度优化。

4　结束语

为了实现云计算中工作流的调度优化，本文提出了一种基于 DAG 分割的工作流调度算法。算法通过任务分割、分割调整及资源分配三个阶段的工作流调度模式，实现了任务执行跨度与执行代价的均衡调度。实验结果证明了所提算法能够以牺牲较小的执行效率开销得到更低的执行代价，其性能优于同类型算法。下一步的研究工作将集中于：将云资源方执行任务的能耗考虑到优化目标中，同时给出工作流执行的截止期限与用户费用约束，建立多约束条件下的多目标优化调度模型，并在新的调度模型下设计相应的 DAG 分割算法，实现多目标调度优化。

参考文献：

[1] Pietri I, Malawski M, Juve G, *et al.* Energy-constrainted provisioning for scientific workflow ensembles [C]// Proc of the 3rd International Conference on Cloud and Green Computing. 2015: 34-41.

[2] Med M, Master H. Auto-scaling to minimize cost and meet application deadlines in cloud workflows [C]// Proc of High Performance Computing, Networking, Storage and Analysis. [S. l.] : IEEE Press, 2014: 1-12.

[3] Rodriguez M A, Buyya R. Deadline based resource provisioningand scheduling algorithm for scientific workflows on clouds [J]. IEEE Trans on Cloud Computing, 2014, 2 (2): 222-235.

[4] Abrishami S, Naghibzadeh M, Epema D H J. Deadline-constrained workflow scheduling algorithms for Infrastructure as a Service Clouds [J]. Future Generation Computer Systems, 2013, 29 (1): 158-169.

[5] Wu C Q, Lin Xiangyu, Yu Dantong, *et al.* End-to-end delay minimization for scientific workflows in clouds under budget constraint [J]. IEEE Trans on Cloud Computing, 2015, 3 (2): 169-181.

[6] Zeng Lingfang, Veeravalli B, Li Xiaorong. ScaleStar: budget conscious scheduling precedence-constrained many-task workflow applications in cloud [C]// Proc of IEEE 26th International Conference on Advanced Information Networking and Applications. Piscataway, NJ: IEEE Press, 2012, 11 (1): 534-541.

[7] Arabnejad H, Barbosa J G. A budget constrained scheduling algorithm for workflow applications [J]. Journal of Grid Computing, 2014, 12 (4): 665-679.

[8] Zheng Wei, Sakellariou R. Budget-deadline constrained workflow planning for admission control [J]. Journal of Grid Computing, 2013, 11 (4): 633-651.

[9] Lee Y C, Zomaya A. Stretch out and compact: workflow scheduling with resource abundance [C]// Proc of IEEE//ACM International Symposium on Cluster, Cloud and Grid Computing. IEEE, 2013: 219-226.

[10] Sakellariou Red, Zhao Hao. A hybrid heuristic for DAG scheduling on heterogeneous systems [C]// Parallel and Distributed Processing Symposium. Proceedings. International. [S. l.] : IEEE Press, 2014: 111-114.

[11] Topcuoglu H, Hariri S, Wu Minyou. Performance-effective and low-complexity task scheduling for heterogeneous computing [J]. IEEE Trans on Parallel & Distributed Systems, 2012, 13 (3): 260-274.